Energy-QoS tradeoffs in J2EE hosting centres

Aeiman Gadafi, Daniel Hagimont*, Laurent Broto and Remi Sharrock

Institut de Recherche en Informatique de Toulouse,
118 route de Narbonne,
31062 Toulouse, France
E-mail: aeiman.gadafi@irit.fr
E-mail: daniel.hagimont@irit.fr
E-mail: laurent.broto@irit.fr
E-mail: remi.sharrock@irit.fr
*Corresponding author

Alain Tchana and Noel De Palma

INRIA Rhone-Alpes,
655 avenue de l'Europe
38334 Montbonnot, France
E-mail: alain.tchana@inria.fr
E-mail: noel.depalma@inrialpes.fr

Abstract: Nowadays, hosting centres are widely used to host various kinds of applications (e.g., web servers or scientific applications). Resource management is a major challenge for most organisations that run these infrastructures. Many studies show that clusters are not used at their full capacity which represents a significant source of waste. Autonomic management systems have been introduced in order to dynamically adapt software infrastructures according to runtime conditions. They provide support to deploy, configure, monitor, and repair applications in such environments. In this paper, we report our experiments in using an autonomic management system to provide resource aware management for a clustered application. We consider a standard replicated server infrastructure in which we dynamically adapt the degree of replication in order to ensure a given QoS while minimising energy consumption.

Keywords: administration; energy management; replication; autonomic computing.

Reference to this paper should be made as follows: Gadafi, A., Hagimont, D., Broto, L., Sharrock, R., Tchana, A. and De Palma, N. (2014) 'Energy-QoS tradeoffs in J2EE hosting centres', *Int. J. Autonomic Computing*, Vol. 2, No. 1, pp.54–72.

Biographical notes: Aeiman Gadafi is a PhD student at the University of Toulouse and researcher in the IRIT Laboratory. His main research topic is power management in distributed infrastructures.

Daniel Hagimont is a Professor at Polytechnic National Institute of Toulouse and a member of the IRIT Laboratory, where he leads a group working on operating systems, distributed systems and middleware. He received his PhD from Polytechnic National Institute of Grenoble in 1993. After a Postdoc at the University of British Columbia, Vancouver in 1994, he joined INRIA Grenoble in 1995. He took his Professor position in Toulouse in 2005.

Laurent Broto received his PhD from the University of Toulouse in 2008. After a Postdoctoral stay at Oak Ridge National Laboratory, TN, USA, he was recruited as an Associate Professor in Computer Science at the Polytechnic National Institute of Toulouse, France.

Remi Sharrock received his PhD in Computer Science from the University of Toulouse in 2010. After a Postdoctoral stay at the National Institute for Research in Computer Science and Control (INRIA) in Bordeaux, he is currently a Teacher-Researcher for the French Ministry of Industry working for the Ecole des mines de Nantes. His main research topic is dynamic reconfiguration in autonomic computing systems.

Alain Tchana received his PhD in Computer Science from the University of Toulouse in 2011. He is currently a Postdoctoral Fellow at the University of Grenoble. His main research topic is autonomic management in cloud infrastructures.

Noel De Palma received his PhD in Computer Science from the Grenoble Institute of Technology in 2001. Since 2002, he has been an Associate Professor in Computer Science at University of Grenoble INP. He is a member of the Sardes research group at INRIA and LIG Laboratory, where he conducts research on autonomic computing.

1 Introduction

Nowadays, medium or large-scale distributed infrastructures such as clusters and grids are widely used to host various kinds of applications (e.g., web servers or scientific applications). The development of *cloud computing* (Buyya et al., 2009) illustrates a general trend towards the emergence of large-scale hosting centres.

Power consumption is becoming a major challenge for most organisations that run these infrastructures. According to the US Environmental Protection Agency, it is estimated that this sector consumed about 61 billion kilowatt-hours (kWh) in 2006 (1.5% of total US electricity consumption) for a total electricity cost of about $4.5 billion. Moreover, energy consumption of these infrastructures is estimated to have doubled between 2000 and 2006 and the development of hosting centres will amplify this tendency.

In clustered infrastructures, a classical structuring pattern is to replicate servers in order to enforce scalability. In this pattern, a given server is replicated statically at deployment time and a frontend proxy acts as a load-balancer and distributes incoming requests among the replicas. Such a design choice induces resource overbooking to face load peaks while guaranteeing quality of service. As there is relatively little difference in power consumption between an idle node and a fully used node, there is a penalty (regarding energy) for keeping an idle node powered on. Moreover, such hosting

infrastructures require large cooling systems which also consume a considerable amount of energy.

Autonomic management systems (Kephart and Chess, 2003) have been proposed as one solution for the management of distributed infrastructures. Such systems can be used to deploy and configure applications in a distributed environment. They can also monitor the environment and react to events such as failures or overloads and reconfigure applications accordingly and autonomously.

In this context, we aim at using an autonomic management system in order to dynamically adapt the degree of replication according to the received load. This adaptation is a mean to dynamically allocate or free machines, i.e., to dynamically turn cluster nodes on – to be able to efficiently handle the load imposed on the system – and off – to save power under lighter load. We implemented such a management policy for a clustered database server in a J2EE application, and evaluated the benefits for energy consumption.

Instead of turning machines on and off which is quite costly and slow, we use an efficient suspension to RAM mechanism to quickly turn the machines in power saving mode.

The rest of the paper is structured as follows: Section 2 presents the context of our work and our motivations. Section 3 describes our approach. Section 4 presents the experiments to evaluate our approach. After an overview of related works in Section 5, we conclude in Section 6.

2 Context

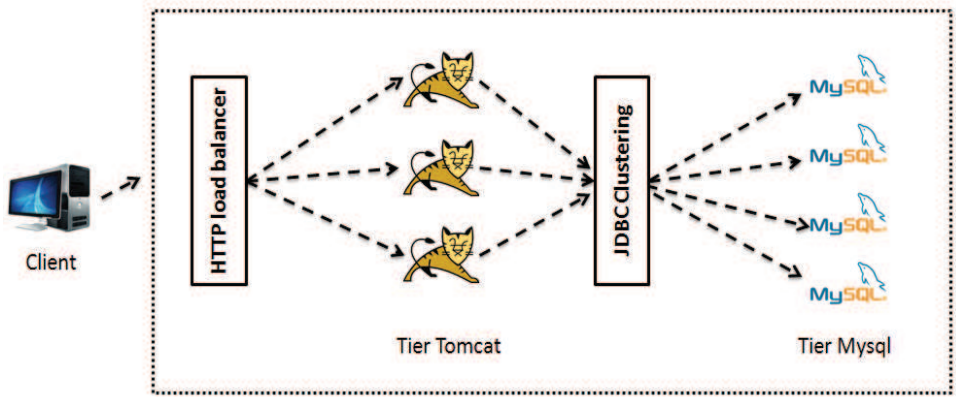
In this section, we first present the application case that we use to illustrate our approach. We then overview the autonomic management system that we used in our experiments.

2.1 Clustered J2EE application

As experimental environment, we made use of the J2EE, which defines a model for developing web applications in a multi-tiered architecture. Such applications are typically composed of a web server (e.g., Apache), an application server (e.g., Tomcat) and a database server (e.g., MySQL). Upon an HTTP client request, either the request targets a static web document, in this case the web server directly returns that document to the client; or it refers to a dynamically generated document, in that case the web server forwards the request to the application server. When the application server receives a request, it runs one or more software components (e.g., Servlets, EJBs) that query a database through a Java database connection driver (JDBC driver). Finally, the resulting information is used to generate a web document on-the-fly that is returned to the web client.

In this context, the increasing number of internet users has led to the need for highly scalable and highly available services. To deal with high loads and provide higher scalability of internet services, a commonly used approach is the replication of servers in clusters. Such an approach (Figure 1) usually defines a particular software component in front of each set of replicated servers, which dynamically balances the load among the replicas. Here, different load balancing algorithms may be used, e.g., random, round-robin, etc.

Figure 1 Clustered J2EE servers (see online version for



In such an architecture, a difficult issue is to find the best degree of replication for each tier, which should be sufficient to tolerate load peaks, but not too high to prevent resource wasting.

2.2 Component-based autonomic management systems

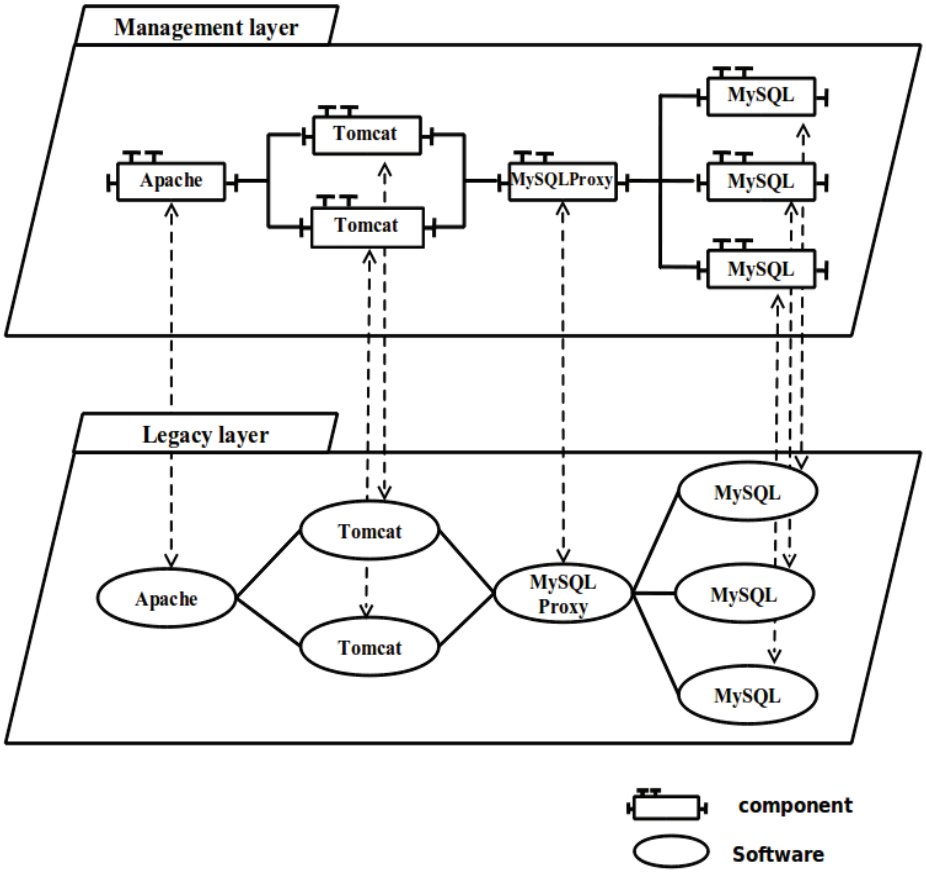
In the area of autonomic computing, many studies have relied on a component model to provide an autonomic system support (Cheng et al., 2004; Hagimont et al., 2006; Oriezy et al., 1999). Component-based management aims at providing a uniform view of a software environment composed of different types of servers. Each managed server is encapsulated in a component and the software environment is abstracted as a component architecture. Therefore, deploying, configuring and reconfiguring the software environment is achieved by using the tools associated with the choose component-based middleware.

In previous projects, we designed and implemented several prototypes of such an autonomic management system (Hagimont et al., 2009; Gadafi et al., 2010; Tchana et al., 2010; Sharrock et al., 2010). In our approach, the component model is used to implement a *management layer* on top of the *legacy layer* composed of the actual managed software (Figure 2).

In the management layer, all components provide a management interface for the encapsulated software, and the corresponding implementation is specific to each software (e.g., the Apache web server). These interfaces are used to control uniformly the components, they allow managing the component's attributes or bindings with other components, and controlling its internal configuration state.

These components interact with remote representatives for configuring the legacy software instances on the remote hosts of the cluster.

Figure 2 Management layer



Relying on this management layer, sophisticated administration programmes can be implemented. However, we observed that the interfaces of a component middleware are too low level and are therefore not appropriate for administrators. This led us to the design of higher level language support for autonomic management policy specification.

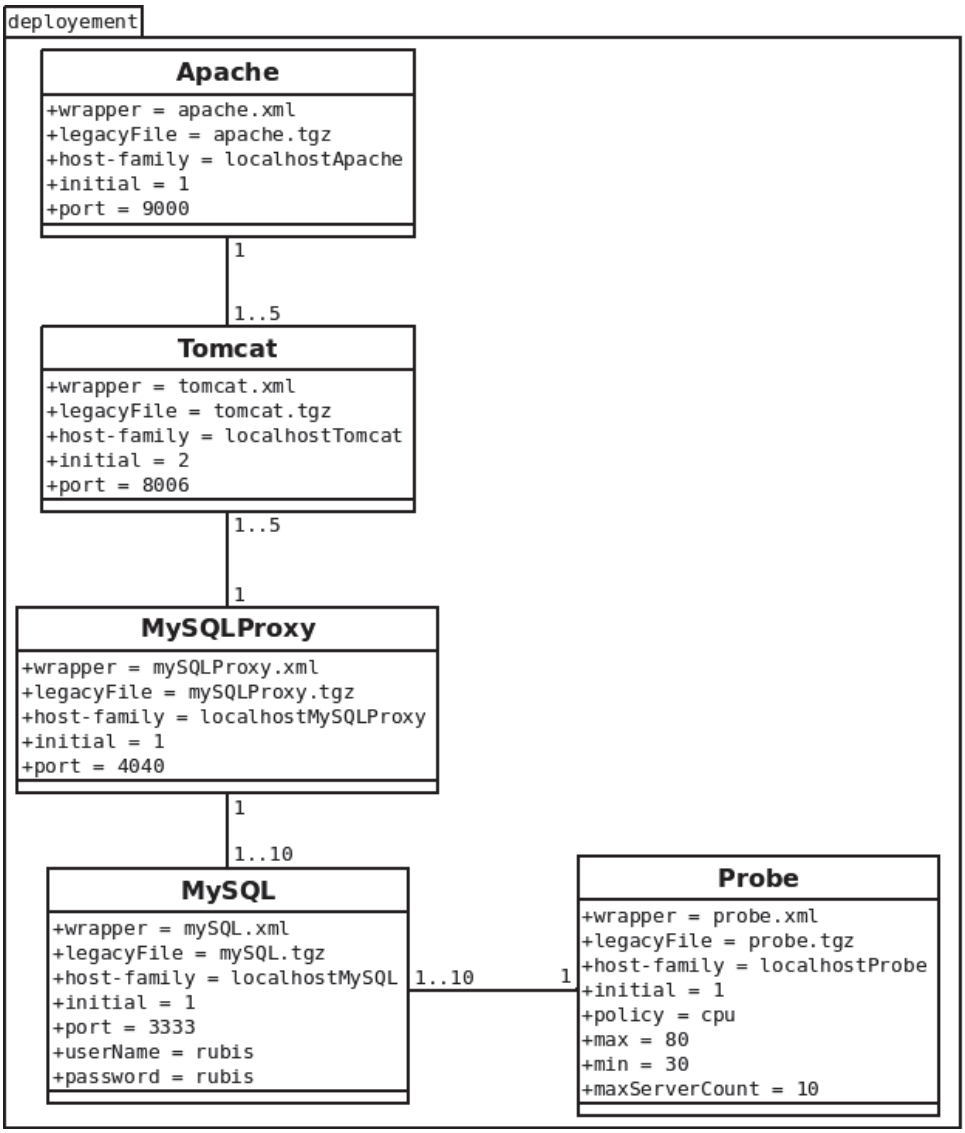
2.3 High level formalisms for policy specification

In order to allow the administration by non-experts, we developed TUNe (Broto et al., 2008), which is a component-based autonomic management system that provides high level formalisms for the description of the software architecture, its deployment and reconfiguration policies. These formalisms are inspired by Unified Modelling Language (UML) and are overviewed in the next sub-sections.

2.3.1 A UML profile for deployment scheme

TUNe introduces a UML profile based on the class diagram for graphically describing deployment scheme. A deployment scheme describes the overall organisation of a software infrastructure to be deployed. At deployment time, the scheme is interpreted to deploy the component architecture. An example is depicted in Figure 3.

Figure 3 Deployment schema for J2EE



Each element of this diagram corresponds to a software which can be instantiated in several component replicas. A link between two elements generates bindings between the components instantiated from these elements. An element includes a set of configuration attributes for the software. Most of these attributes are specific to the software, but few attributes are predefined by TUNe and used for deployment, for example the *initial* attribute which indicates the number of instances that have to be deployed initially, thus each component may be replicated by adapting this attribute.

2.3.2 A WDL

Upon deployment, the above schema is parsed and for each element, the initial number of components are created. These components implement some wrappers for the deployed software, which provide control over the software. Each wrapper component is an instance of a generic wrapper which is actually an interpreter of a Wrapping Description Language (WDL) specification.

A WDL description defines a set of methods that can be invoked to configure or reconfigure the wrapped software. The workflow of methods that have to be invoked in order to configure and reconfigure the overall software environment is defined thanks to an formalism introduced in Section 2.3.3.

Usually, a WDL specification provides *start* and *stop* methods for controlling the activity of the software, and a *configure* method for reflecting the values of the attributes (defined in the UML deployment schema) in the configuration files of the software. Note that the values of these attributes can be modified dynamically at runtime. Other methods can be defined according to the specific management requirements of the wrapped software.

The methods described in a WDL specification are implemented in Java. Most of these methods are generic and can therefore be reused, e.g., methods which access a configuration file in a particular format.

2.3.3 A UML profile for (re)configuration procedures

TUNe introduces a UML profile based on state diagrams. These diagrams are used to define workflows of operations that have to be performed during runtime for reconfiguring the managed environment. Reconfigurations are triggered by events that start the different workflows. An event can be generated by a specific monitoring component, or even by a wrapped legacy software itself if it includes its own monitoring functions. An operation in a state diagram can assign an attribute or a set of attributes of components, create, deploy and destroy components, or can invoke a method or a set of methods on components.

The deployment scheme implicitly defines a designation scheme for addressing the deployed component instances. When an event is raised by a component (e.g., a probe), the *this* keyword identifies this component and the designation scheme allows navigation in the deployed architecture for invoking methods on components.

2.4 Motivations

One important autonomic management policy we target in this paper is the minimisation of energy consumption while meeting the end user needs (i.e., keeping response time

acceptable). This management system aims at autonomously increasing/decreasing the number of replicated resources used by the application when the load increases/decreases. This has the effect of efficiently maximising resource utilisation (i.e., no resource overbooking) and therefore minimising the overall energy consumption. Such a policy would aim at:

- guaranteeing the performance of the application by adapting its size as needed
- reducing energy consumption by shutting down the unused nodes.

3 Autonomic approach to trading energy and QoS off

In this section, we present two management policies that we implemented in order to manage energy consumption in a clustered J2EE architecture. These policies apply to the database tier in the J2EE architecture.

3.1 Technical issues

3.1.1 Management of the replicated database tier

The load-balancer among replicated databases is MySQLProxy. The database load, arriving from the web server is distributed by MySQLProxy to the DB replicas that can be added and removed based on workload variations. In order to implement the integration of a new database backend in a pool of replicas, we have to consider two cases, according to whether the database can be modified or not.

- *Read-only access:* In this simple case, all nodes (on or off) contain the same database content and do not require any special care when integrating a new database server in the database pool.
- *Read-write access:* In this case, we have to take into account the current state of the replicated database. The technique we use leverages the logging facilities of MySQLProxy. For each new node activation, we execute a reconciliation operation on the node, thus bringing it in the same state as all the database replicas. To implement the reconciliation, the log file is used to replay all the SQL statements that have been recorded since the last synchronisation state. This is a relatively fast operation given the fact the read/write ratio is high; however it depends on the time between state synchronisations and the number of writes during this time.

3.1.2 Suspension to RAM

Turning machines off, and especially on is quite costly. Indeed, we measured that such an operation takes about 45 seconds on the average. Instead, we rely on suspension to RAM, which allows to suspend and resume the activity of a machine at a low cost (about four seconds on the average for resuming a machine) while saving as much energy as if it were turned off (Talebi and Way, 2009). Suspend-to-RAM stores information on the system configuration, the open applications, and the active files in main memory (RAM), while most of the system's other hardware is turned off. When a machine is suspended, only the RAM and the network device are powered on.

3.2 Autonomic management policies

We aim at minimising energy consumption of the cluster, while ensuring acceptable QoS, and being transparent to the end-users. As mentioned before, the management system autonomously increases/decreases the number of replicated resources used by the application when the load increases/decreases.

The J2EE architecture is initially deployed with one database server (MySQL). We use probes to monitor some performance metrics on MySQL server nodes. These probes periodically collect the informations on all the nodes where MySQL servers are deployed. They compute a moving average of the collected data in order to remove any artefact. They finally compute an average across all nodes, so as to observe a general load indication of the whole replicated server pool. According to defined thresholds, the probe will eventually generate an event to add or remove a replica.

We defined two different policies: the first provides the maximum QoS, while the second allows the degradation of the QoS to save more energy.

- *First policy, maximum QoS*: In this policy, we aim at minimising the consumed energy while maintaining the maximum QoS that can be given by the system.

  Addressing the energy and QoS tradeoff, we give priority to performance by distributing the current workload onto a sufficient number of machines. This number must be enough to serve end-users without effecting the QoS, while turning off the unused machines to save energy.

  To ensure the best QoS, the MySQL servers should never be saturated, thus we monitor their status (i.e., CPU usage) and add more DB replicas to prevent saturation.

  From the end-users point of view, since there is no degradation of service, the process of adding or removing a replica is completely transparent.

- *Second policy, degraded QoS*: Instead of looking at MySQL servers' status, we are interested here in monitoring the QoS indicators, especially the response time.

  If we allow application performance to be somewhat degraded, much greater energy savings are possible. In this policy, the application's performance is measured by a service level agreement (SLA) related to response time. This allows to reduce the amount of resources allocated to the applications to the point where the SLA goals are just being met. More precisely, the SLA specifies the range where the response time must be maintained. This range is obviously higher than the response time that is obtained with the previous policy (response time is degradated), and it allows managing the application with fewer DB servers. Figure 9 shows the occupancy rate of machines during the experiments with the two implemented policies. If we compare the occupancy rate of the static configuration with three database servers, and the occupancy rate in the dynamic cases, we observe that the CPU-based policy reduces the occupancy rate by 30%, while the latency-based policy reduces the occupancy rate by 47%.

3.3 Implementation with TUNe

Figure 3 shows the deployment schema for the J2EE architecture, which includes a frontend web server (*Apache*), an application server (*Tomcat*), a database load-balancer (*MySQLProxy*) and a database server (*MySQL*). The *MySQL* server is monitored by a Probe programme which is parameterised by different attributes:

- *max*: this parameter specifies the maximum threshold of the probed property (e.g., CPU usage or latency) that can be reached before triggering the addition of a replica
- *min*: this parameter specifies the minimum threshold of the probed property that can be reached before triggering the removal of a replicas
- *maxServerCount*: this parameter specifies the maximum number of replica
- *policy*: this parameter specifies the used policy, i.e., the probed property (CPU usage or latency).

The probe will eventually generate an event to add or remove a replica, as we describe in the next sections.

3.3.1 Adding a new replica

To add a new replica, we must ensure that the resulting load is higher than the value of the *min* parameter to prevent removing the replica just after adding it. The condition to add a new replica can therefore be formulated as:

$$\left(\frac{CL}{(N+1)} > min\right) \text{ and } \left(N < maxServerCount\right)$$

where *CL* is the current load, and *N* is the number of currently used servers.

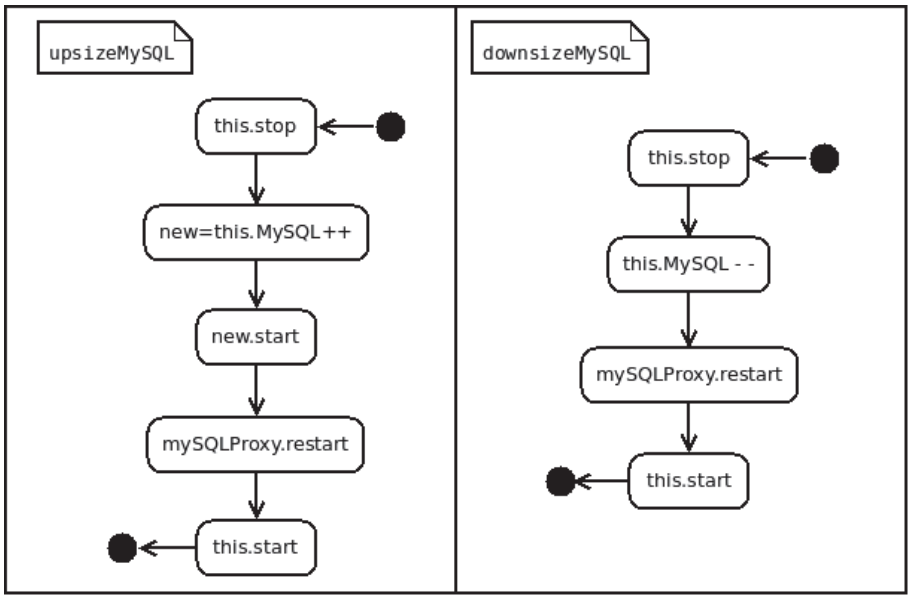
When this event is generated, TUNe allocates a node from a list of available nodes for this server, turns the allocated node on via a Wake-on-LAN notification, and applies the reconfiguration diagram corresponding to the addition of a new server when the node is ready.

Figure 4(a) shows the reconfiguration diagram for adding a new MySQL server. The event (*upsizeMySQL*) is generated by a *Probe* component instance, therefore the this variable is the name of this *Probe* component instance. Then:

- *this.stop* invokes the *stop* method on the probing component to prevent the generation of multiple events.
- *new=this.MySQL++* creates a new MySQL instance. Note that in order to designate the component on which an operation should be performed, a dotted syntax allows to navigate through the component architecture, e.g., here to follow the link between the *Probe* and the *MySQL* elements. The ++ operator increases by one the number of instances of the *MySQL* element.
- *new.start* invokes the *start* method on the newly created *MySQL* component instance.

- *mysqlProxy.restart* restarts the database load-balancer to take into account the newly created *MySQL* server.
- *this.start* restarts the probe.

Figure 4 State diagrams for adding/removing a MySQL server



(a) (b)

3.3.2 Removing a replica

To remove a replica, we must ensure that the resulting load is lower than the value of the *max* parameter to prevent adding a replica just after removing it. So the condition to remove a replica can be formulated as:

$$\left(\frac{CL}{(N-1)} > max \right) \quad \text{and} \quad (N > 1)$$

When this event is generated, TUNe applies the reconfiguration diagram which removes a replica and frees the node (with a suspend-to-ram).

Figure 4(b) shows the reconfiguration diagram for removing a MySQL server.

4 Evaluation

In this section, we present the evaluation of our solution. First we introduce the application we deployed (an auction website modelled over eBay), the software environment on which we deployed the application (the J2EE platform) and the hardware (a French grid). Then we describe the test configuration and how we evaluate the energy consumption.

Testbed application

The evaluation has been realised with RUBiS (Amza et al., 2002), a J2EE application benchmark based on servlets, which implements an auction site modelled over eBay. It defines 26 web interactions, such as registering new users, browsing, buying or selling items. RUBiS also provides a benchmarking tool that emulates web client behaviours and generates a tunable workload. This benchmarking tool gathers statistics about the generated workload and the web application behaviour.

Software environment

The nodes run version 2.6.30 of the Linux kernel. The J2EE application has been deployed using open source middleware solutions: Jakarta Tomcat 6.0.20 for the web and servlet servers, MySQL 5.1.36 for the database servers, MySQLProxy 0.7.2 for the database load-balancer, Apache 2.2.14 for the application server load-balancer. We used RUBiS 1.4.3 as the running J2EE application. These experiments have been realised with Sun's JVM JDK 1.6.0.05. We used the MySQL connector/J 5.1 JDBC driver to connect the database load-balancer to the database servers.

Hardware environment

The experimental evaluation was carried out using the Grid'5000 experimental testbed.[1] The experiments required up to 14 nodes: one node for TUNe management platform, one node for web server (*Apache*), six nodes for servlet servers (*Tomcat*), one node for database load-balancer (*MySQLProxy*), up to three nodes for database servers (*MySQL*), two nodes for RUBiS client emulators (which emulate up to 3,000 clients). The number of nodes actually used during these experiments varies, according to the dynamic changes of the workload, resulting in the dynamic resizing of the application. All the nodes are connected through a 100 Mbps ethernet LAN to form a cluster.

Test configuration

In this evaluation, we provide measurements for the database replicated tier only. The RUBiS benchmark is configured to send read-only queries. The parameters that control adding/removing servers are shown in Table 1.

Table 1 Test configuration

Parameter name	*CPU-based policy*	*Latency-based policy*
max	80%	6,000 milliseconds
min	30%	20 milliseconds

Evaluation scenario

We aim at showing that dynamic allocation and deallocation of nodes in response to workload variations allows energy saving.

In our benchmark, the load increases progressively up to 3,000 emulated clients: 50 new clients every 30 seconds. We configured RUBiS client to run for 31 minutes, so

the total time of the experiments is about one hour. During the experiment, the probe monitors the MySQL server nodes.

The J2EE architecture is initially deployed with one database server (MySQL) and TUNe reacts to the load variation by allocating and freeing nodes, as described in Section 3.3

We compare the QoS (e.g., latency) variable and the energy consumption (e.g., occupancy rate of machines) in two situations:

- Static configuration of the J2EE architecture. We measure the performance (regarding QoS and energy consumption) with one, two and three database servers. Fewer servers will save energy but with a degradation of QoS. More servers will optimise QoS, but with energy wasting.

- Dynamic configuration of the J2EE architecture. TUNe is used to dynamically adapt the number of database servers as described above. Therefore, quality of service is guaranteed without wasting energy. We then compare the two policies described in Section 3.2.

Energy evaluation

The first experiment we conducted was to measure the energy consumption of one node according to its load. To do this we used a framework that collects energy usage information in Grid'50002. This measurement is shown in Table 2.

Table 2 Energy consumption for one node

CPU usage (%)	*Energy consumption (watts)*
0%	204
50%	211
100%	224

From these measurements, we can see that keeping a machine on when it is not used has a high cost. This means that powering off machines is the most effective way to save energy.

4.1 Results

4.1.1 Static configuration

In a first evaluation, we measured the latency and the CPU usage of the J2EE application when it is statically configured with one, two and three MySQL servers. These measurements are given in Figures 5 and 6.

We observe that when we use one database server, it becomes saturated between times 750 and 2,950 (Figure 5). This saturation has an impact on the quality of service as shown by Figure 6. Indeed, when the server is saturated, the latency increases at the same time. We observe the same behaviour when we used two servers (between times 1,500 and 2,200). The configuration with three database servers can maintain the quality of service (the latency is not increasing) for all the experiment's time.

Figure 5 CPU usage in static configuration (see online version for colours)

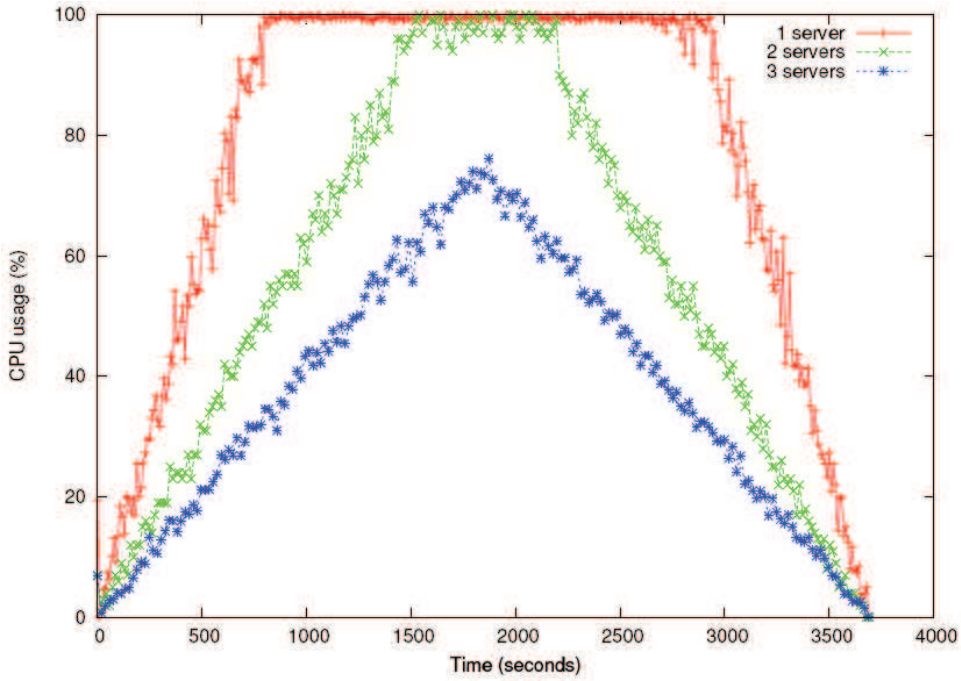
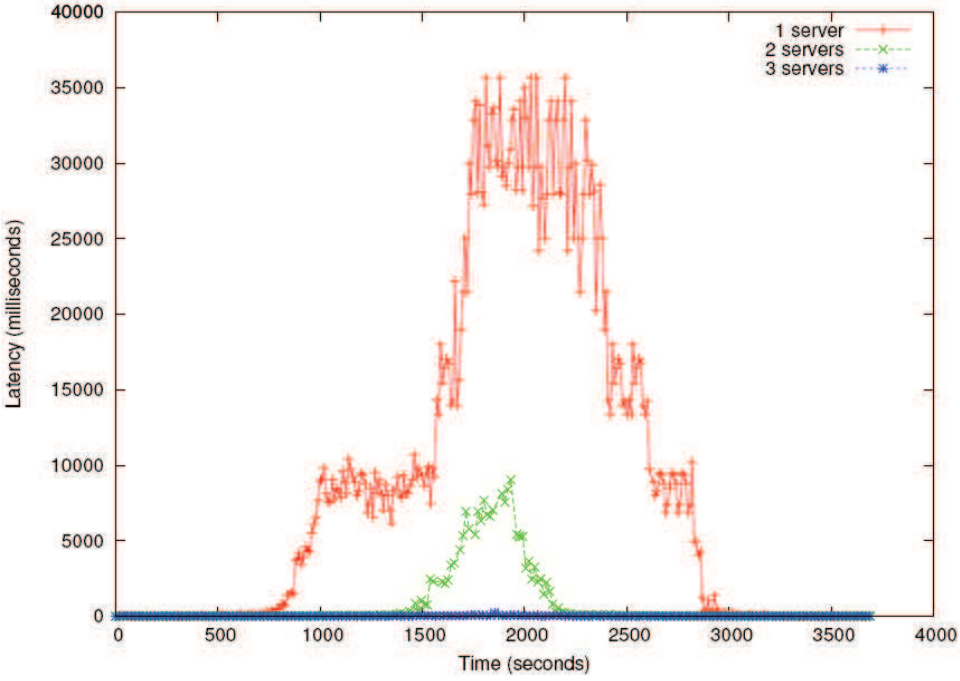


Figure 6 Latency in static configuration (see online version for colours)



For the static configuration mode, we can see that maintaining the quality of service, particularly during the peak load, requires three database servers, but these three servers are not required during the whole experiment. Indeed, regarding the CPU usage in Figure 5 and the latency in Figure 6, only one server is required from time 0 to 600 and time 3,200 to 4,000. Only two servers are also required from times 600 to 1,400 and times 2,400 to 3,200. Therefore, up to two servers could be switched off to save energy.

4.1.2 Dynamic configuration

With TUNe, we dynamically turn cluster nodes on – to be able to handle load peaks – and off – to save power under lighter load. We only use nodes (and consume energy) when needed.

Figure 7 shows the CPU load and the latency when using TUNe configured with the CPU-based policy. To ensure the best QoS we do not allow server saturation. We start with only one MySQL server, when this server approaches the saturation point (when the CPU load reaches 80%), TUNe adds a new server to the architecture. This occurs twice in our experiment: at times 690 and 1,400. We observe that using this policy ensures the maximum quality of service regarding the latency, which is relatively stable during all the experiment.

Figure 8 shows the CPU load and the latency when using TUNe configured with the latency-based policy. Unlike the previous case, we allow QoS to be degraded in order to save more energy. We observe this performance degradation regarding the latency. When the latency reaches the predefined threshold (6,000 milliseconds) TUNe adds a new server to the architecture. This occurs twice at times 1,030 and 1,930.

Figure 7 CPU-based policy (see online version for colours)

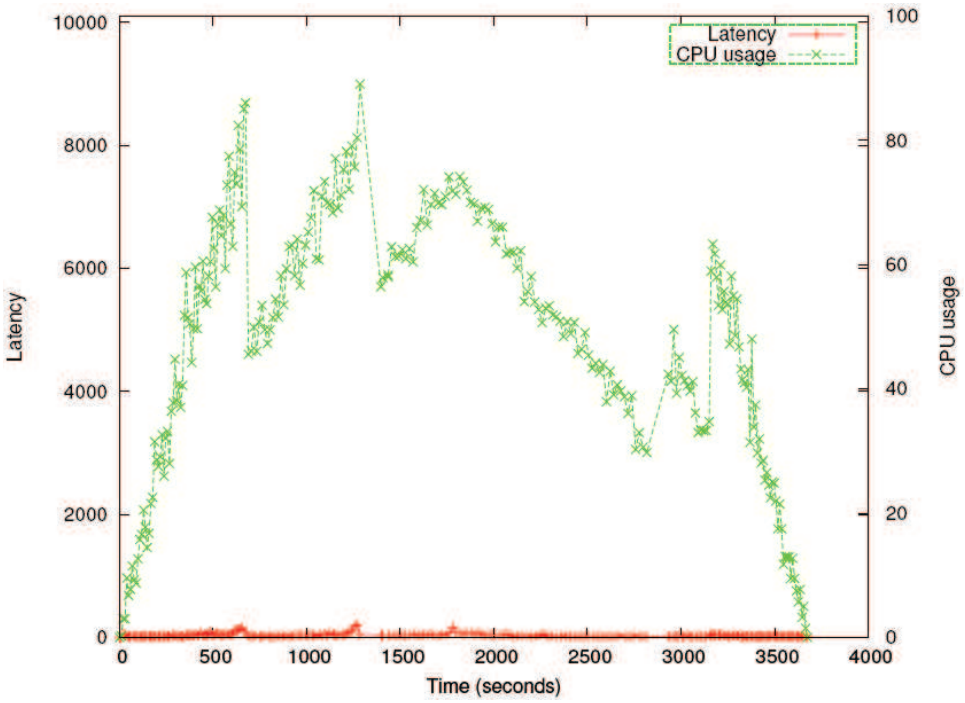


Figure 8 Latency-based policy (see online version

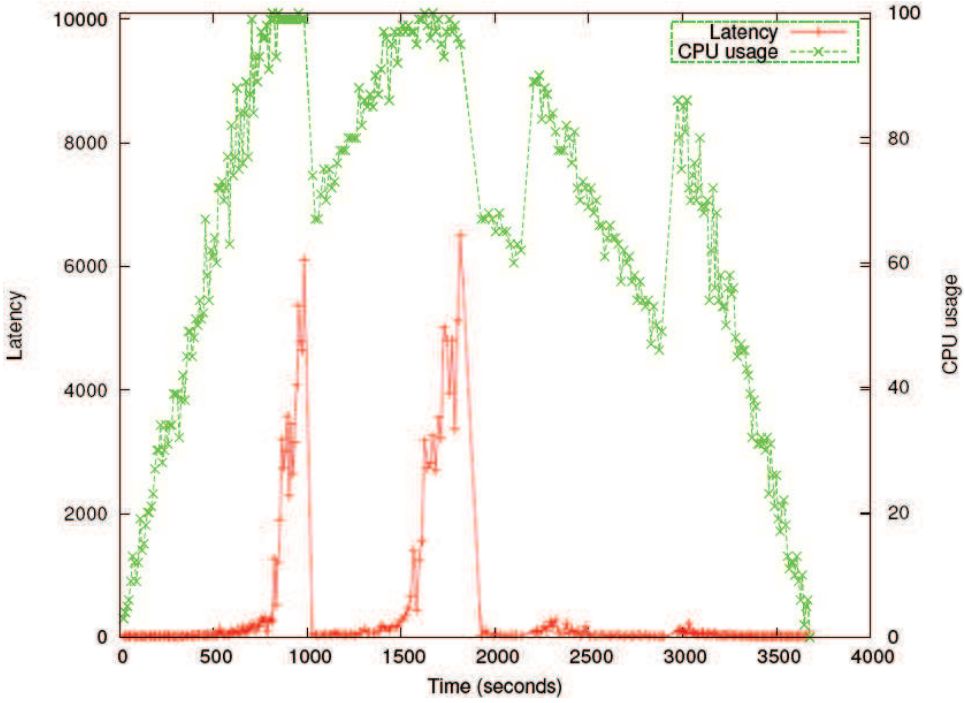


Figure 9 Static configuration (see online version for colours)

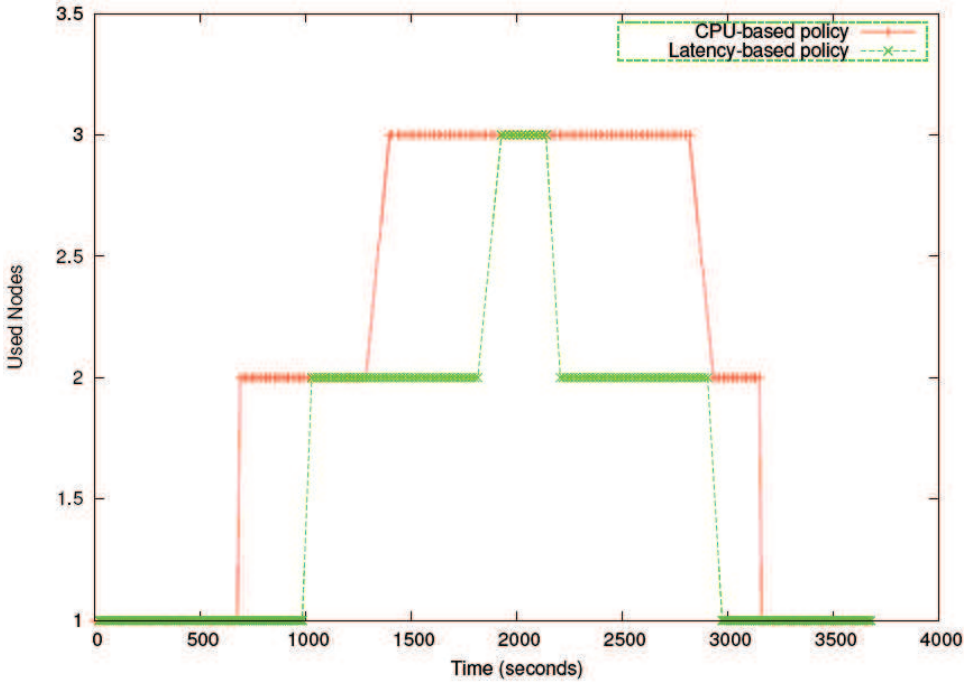


Figure 9 shows the occupancy rate of machines during the experiments with the two implemented policies. If we compare the occupancy rate of the static configuration with three database servers, and the occupancy rate in the dynamic cases, we observe that the CPU-based policy reduces the occupancy rate by 30%, while the latency-based policy reduces the occupancy rate by 47%.

5 Related work

Many works have addressed the issue of power management. Most of them have focussed on energy management for electronic devices powered by electric battery, and few have addressed this issue in grid or cluster infrastructures.

Many research works which aim at managing energy for a single processor system can be used to optimise energy consumption independently on each node of a cluster. It includes optimisations of the use of the processor, memory or input/outputs (disk or network).

Most of the research that focuses on cluster-wide energy management deals with resource allocation. We can mention some examples of such works:

- *Load balancing*: in this category, we can cite the work of Pinheiro et al. (2001) who developed an algorithm which makes load balancing decisions by considering both the total load imposed on the cluster and the power and performance implications of turning nodes off.
- *Virtualisation*: in this category, we can cite the work of Hermenier et al. (2006) who developed a system which relies on virtual machine migration for transparently relocating any application in the cluster. The placement policy takes into account the CPU and memory usages, in order to concentrate the workload on fewer nodes of the cluster, thus allowing unused nodes to be shutdown. We are currently cooperating with them to integrate our autonomic management system with their work.
- *Simulation*: we can here cite the work of Khargharia et al. (2008). They present a theoretical framework and methodology for autonomic power and performance management in data centres. They rely on simulation to apply their approach to two case studies, a multi-chip memory system and a high performance server cluster.

Our work is orthogonal to these contributions. While most of the works made in this domain is specific to energy management, our autonomic computing approach is generic, as it can be used to define any management policy for any distributed software architecture. The field of energy management was not previously addressed by our autonomic management system, but the experiments reported in this paper show that our approach can by used to define a specific energy management autonomic policy.

The closest work to our is that of Das et al. (2008) who proposed a multi-agent approach for managing power and performance in server clusters by turning off servers under low-load conditions. Instead of relying on components and architectures, their autonomic system follows a multi-agent paradigm. Our approach differs from this work in several respects. First, multi-agent paradigms are programmed to manage specifically the dynamicity of systems. Our approach focuses more on managing in a generic way legacy systems that are intrinsically static (for example web servers or database servers). Second, we differ from previous approaches that use multicriteria utility functions in that

we employ high level policies descriptions to dynamically reconfigure the managed systems. This allows multiple goals definitions and goals combinations like repairing or dynamic sizing. Because we offer a complete implementation of an autonomic manager that takes into account multiple goals, we show in this paper that both the application performance metrics and hardware power consumption metrics can be used to optimise the system. Finally, these reconfiguration policies are externalised (not related to a specific application but more generic and high level) and can be applied to other applications or changed easily, which is not the case with a multi-agent system.

6 Conclusions and future work

Nowadays, medium or large-scale distributed infrastructures such as clusters and grids are widely used to host various kinds of applications. Power consumption has become a major challenge for most organisations that run these infrastructures. Many studies show that they are not used at their full capacity and that there are therefore a significant source of wasted power. Autonomic management systems have been recognised as a convenient solution for management of distributed infrastructures.

The experiments that we conducted show that the autonomic computing approach can be used for energy management in a distributed infrastructure. This approach meets the needs of energy aware computing, as it can minimise power consumption without affecting the performance of the system. In our experiments, we were able to obtain for a typical web application benchmark a reduction of the power consumption between 30% and 47% according to the used policy.

This paper reported on preliminary work. In the near future, we aim at evaluating much deeply our prototype through more elaborated power management policies, which would include other parameters, for example, network traffic information. We also wish to integrate virtualisation techniques in our prototype, as it would enable transparent process (VM) migration between hardware nodes.

Acknowledgements

The work reported in this paper benefited from the support of the French National Research Agency through project SelfXL (ANR-08-SEGI-017-04).

References

Amza, C., Cecchet, E., Chanda, A., Cox, A., Elnikety, S., Gil, R., Marguerite, J., Rajamani, K. and Zwaenepoel, W. (2002) 'Specification and implementation of dynamic web site benchmarks', *5th Workshop on Workload Characterization*, pp.3–13.

Broto, L., Hagimont, D., Stolf, P., Depalma, N. and Temate, S. (2008) 'Autonomic management policy specification in Tune', *Proceedings of the 2008 ACM Symposium on Applied Computing, SAC '08*, Fortaleza, Ceara, Brazil, pp.1658–1663, ACM, New York, NY, USA, ISBN 978-1-59593-753-7, DOI 10.1145/1363686.1364080 [online] http://doi.acm.org/10.1145/1363686.1364080 (accessed 16 October 2012).

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I. (2009) 'Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility', *Computer Future Generation Systems*, Vol. 25, No. 6, pp.599–616.

Cheng, S.W., Huang, A.C., Garlan, D., Schmerl, B. and Steenkiste, P. (2004) 'Rainbow: architecture-based self-adaptation with reusable infrastructure', *International Conference on Autonomic Computing*, pp.276–277.

Das, R., Kephart, J., Lefurgy, C., Tesauro, G., Levine, D. and Chan, H. (2008) 'Autonomic multi-agent management of power and performance in data centers', *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp.107–114.

Gadafi, A., Broto, L., Sayah, A., Hagimont, D. and Depalma, N. (2010) 'Autonomic energy management in a replicated server system', *6th IEEE International Conference on Autonomic and Autonomous Systems.*

Hagimont, D., Bouchenak, S., De Palma, N. and Taton, C. (2006) 'Autonomic management of clustered applications', *IEEE International Conference on Cluster Computing*, pp.1–11.

Hagimont, D., Stolf, P., Broto, L. and Palma, N. (2009) 'Component-based autonomic management for legacy software', in Zhang, Y., Yang, L.T. and Denko, M.K. (Eds.): *Autonomic Computing and Networking*, pp.83–104, Springer, USA, ISBN 978-0-387-89828-5 [online] http://dx.doi.org/10.1007/978-0-387-89828-5_4 (accessed 16 October 2012).

Hermenier, F., Loriant, N. and Menaud, J.M. (2006) 'Power management in grid computing with Xen', *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, pp.407–416.

Kephart, J.O. and Chess, D.M. (2003) 'The vision of autonomic computing', *IEEE Computer Magazine*, Vol. 36, No. 1, pp.41–50.

Khargharia, B., Hariri, S. and Yousif, M. (2008) 'Autonomic power and performance management for computing systems', *Cluster Computing*, Vol. 11, No. 2, pp.167–181.

Oriezy, P., Gorlick, M., Taylor, R., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. and Wolf, A. (1999) 'An architecture-based approach to self-adaptive software', *IEEE Intelligent Systems*, Vol. 14, No. 3, pp.54–62.

Pinheiro, E., Bianchini, R., Carrera, E. and Heath, T. (2001) 'Load balancing and unbalancing for power and performance in cluster-based systems', *Workshop on Compilers and Operating Systems for Low Power*.

Sharrock, R., Monteil, T., Stolf, P., Hagimont, D. and Broto, L. (2010) 'Non-intrusive autonomic approach with self-management policies applied to legacy infrastructures for performance improvements', *International Journal of Adaptive, Resilient and Autonomic Systems IGI Global*, Vol. 2, No. 2, pp.1–20.

Talebi, M. and Way, T. (2009) 'Methods, metrics and motivation for a green computer science program', *40th ACM Technical Symposium on Computer Science Education*, pp.362–366.

Tchana, A., Temate, S., Broto, L. and Hagimont, D. (2010) 'Autonomic resource allocation in a J2EE cluster', *1st International Conference on Utility and Cloud Computing*.

Notes

1 An initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr).